NCC: Natural Concurrency Control for Strictly Serializable Datastores by Avoiding the Timestamp-Inversion Pitfall

Haonan Lu,★
Shuai Mu,♣ Siddhartha Sen,♣ Wyatt Lloyd♠

★University at Buffalo  ♣Stony Brook University
♣Microsoft Research   ♠Princeton University
Transactional Datastores

User

Clients

Tx Logic
Read A, B
Write C
Read D

Servers

Request

User Ops
Results

User Ops
Results

User Ops
Results

User Ops
Results

User Ops
Results
Transactional Datastores

Clients

User

User Ops → Results

User Ops → Results

Servers

Concurrency control

Determines consistency guarantees of the system

By controlling how requests are executed
Strict Serializability

• Transactions take effect in a total order
  ➢ Serializable: requests do not interleave

• Respects the real-time ordering
  ➢ If \( tx_1 \) ends before \( tx_2 \) starts, then \( tx_1 \) must be ordered before \( tx_2 \) in the total order
Strict Serializability Is Costly

- Expensive mechanisms
  - Extra messages, locking, excessive aborts
  - Degrade system performance
- These costs are unnecessary for naturally consistent transactions
Natural Consistency

• Transaction requests arrive in an order that is already strictly serializable

• Prevalent in datacenter workloads
  - Many are reads: Interleaving is okay
  - Many are short: Interleaving is less likely
  - Many arrive in a real-time order: \( tx_1 \) ends before \( tx_2 \) starts, then \( tx_1 \)’s requests must arrive before \( tx_2 \)’s
Executing **naturally consistent** transactions simply in the order they arrive at servers naturally satisfies strict serializability.
NCC: Natural Concurrency Control

• Guarantees strict serializability

• Leverages natural consistency

• Achieves minimal costs in common cases
  
  ➢ One-round latency, lock-free, non-blocking execution
Three Pillars of Design

• Non-blocking execution
• Timestamp-based consistency checking
• Decoupled response management
Non-blocking Execution

• Client pre-assigns timestamps, e.g., physical time
• Requests executed in the order they arrive
• Refine timestamps to match the arrival/execution order
• Immediately visible to subsequent transactions
• Responses are buffered, and sent when safe
Non-blocking Execution Example

tx₁ = \{\text{read A, write B}\}

tx₂ = \{\text{read B, write A}\}
Non-blocking Execution Example

tx₁ = \{\text{read } A, \text{ write } B\}

tx₂ = \{\text{read } B, \text{ write } A\}

Buffered responses:

tx₁.ᵣ₁A ←-- A₀, [0, 5]

tx₁.w₁B ←-- “done”, [5, 5]

tx₂.ᵣ₂B ←-- B₁, [5, 8]

tx₂.w₂A ←-- “done”, [8, 8]
Safeguard

- Timestamp-based consistency checking
- Ensures a total order
  - $[t_w, t_r]$ represents the time range where a request is valid
  - $[t_w, t_r]$ pairs represent the arrival/execution order
  - The intersection of $[t_w, t_r]$ pairs is a serialization point
Safeguard Example

Returned responses:

\[ \text{tx}_1.r_1A \leftarrow A_0, \quad [0, 5] \]
\[ \text{tx}_1.w_1B \leftarrow \text{"done"}, \quad [5, 5] \]
\[ \text{tx}_1.t_{\text{commit}} = 5 \]
\[ \text{tx}_2.r_2B \leftarrow B_1, \quad [5, 8] \]
\[ \text{tx}_2.w_2A \leftarrow \text{"done"}, \quad [8, 8] \]
\[ \text{tx}_2.t_{\text{commit}} = 8 \]
Timestamp-Inversion Pitfall (TIP)

• Fundamental correctness pitfall in timestamp-based strictly serializable techniques

• Timestamps fail to guard against a total order that violates the real-time ordering between transactions in subtle cases
Example of Timestamp Inversion

Execution order is total:
\[ tx_2 \xrightarrow{\text{exe}} tx_3 \xrightarrow{\text{exe}} tx_1 \]

Incorrectly inverts \( tx_1 \xrightarrow{\text{rto}} tx_2 \)

“Alice incorrectly sees \( \text{Ph}_1 \)”

TIP is subtle: \( tx_3 \) interleaves with non-conflicting \( tx_1 \) & \( tx_2 \)

TIP is fundamental: affects various types of transactions in multiple prior systems
Response Timing Control (RTC)

• Control when to send decoupled responses
• Disentangle the subtle interleaving in transactions’ real-time order
• No interference with non-blocking execution
RTC Avoiding TIP

tx₃ arrives before
tx₁ is responded before
Bob is notified before
tx₂ arrives

Execution order is total: 

\[ tx₃ \xrightarrow{exe} tx₁ \xrightarrow{exe} tx₂ \]

Respects \[ tx₁ \xrightarrow{rto} tx₂ \]

“Alice sees \text{Ph}_0\text{, not Ph}_1”
Architecture & Protocol Overview

Client

Execute Phase
- Tx Logic
- Timestamp t
- RO Protocol
- Read A, B
- Write B

Commit Phase
- Safeguard
- Smart retry

Server A
- Non-blocking Execution
- Resp Q (RTC)
- Read A, t
- Res, \([t_w, t_r]\)
- Commit
- Replicated

Server B
- Replicated

User
- Ops
- Results

Backup Coord.
- Replicated

Replicated
Implementation and Evaluation

• Built on Janus’s framework [OSDI ’16]

• Baselines
  - Strictly serializable techniques, e.g., OCC and 2PL
  - Serializable protocols, e.g., MVTO (performance upper bound)

• Workloads
  - Synthetic Facebook-TAO and Google-F1 (read-dominated, one-shot)
  - TPC-C (many writes, multi-shot)
  - Varying write fraction in Google-F1 (write-intensive, one-shot)
### Latency-Throughput, Google-F1

<table>
<thead>
<tr>
<th>Throughput (K Txn/s)</th>
<th>Read Latency (ms) Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>150</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
</tr>
<tr>
<td>250</td>
<td>10000</td>
</tr>
</tbody>
</table>

- **2PL-wound-wait**
- **2PL-no-wait**
- **OCC**
- **NCC**
- **MVTO**

- **O.P.**

> 2x
Conclusion

• NCC: Natural Concurrency Control
  ➢ Minimal-cost, strictly serializable technique, leveraging natural consistency

• Timestamp-inversion pitfall
  ➢ Correctness violation in timestamp-based strictly serializable techniques

• Implementation and evaluation of NCC for datacenter workloads
  ➢ Significantly outperforms strictly serializable solutions
  ➢ Closely matches the performance of serializable techniques

Thank you